95-702 Distributed Systems

Project 2

Assigned: Friday, September 25, 2015

Due: Midnight, Friday, October 9, 11:59pm

Principles

One of our primary objectives in this course is to make clear the fundamental distinction between functional and nonfunctional characteristics of distributed systems. The functional characteristics describe the business or organizational purpose of the system. The non-functional characteristics affect the quality of the system. Is it fast? Does it easily interoperate with others? Is it reliable and secure?

In this project, we attempt to illustrate several nonfunctional characteristics of distributed systems. One such characteristic is reliability. UDP, for example, is considered an unreliable protocol since it does nothing to ensure that packets arrive at their destination. Does that imply that it is of no use? No! One very important UDP application is the Domain Name Service (DNS). In the case of DNS, speed and simplicity may trump reliability. In Task 1, we build a small client and server that use UDP to communicate – a sort of trivial DNS. In Task 2, we add a measure of reliability to UDP with positive acknowledgements and retransmission – a sort of trivial TCP. It's interesting to see how a reliable system can be built upon an unreliable network.

Also, in Task 2, we introduce the proxy design pattern. This is a common pattern in distributed systems. The proxy design pattern is one example of the very sound engineering principle that we should strive to separate concerns. We see the same proxy pattern used in web services, RPC, and RMI. These topics will be visited carefully later in the course.

In Task 3, our focus is on security – a major concern of most distributed system designers. We will build a secure system using symmetric key cryptography. The encryption occurs just above the TCP layer and is provide by the Tiny Encryption Algorithm (TEA). User authentication with ID's and passwords is done in the application layer. This is a common theme in modern systems. User names and passwords are passed over an encrypted, but otherwise insecure, channel.

Be sure to spend some time reflecting on the functional and nonfunctional characteristics of your work. There will be questions on the exams concerning these characteristics. You should be able to demonstrate a nuanced comprehension of

course content and be able to explain the technical aspects in relation to potential real world applications.

For each task below, you must submit screenshots that demonstrate your programs running. These screenshots will aid the grader in evaluating your project.

Documenting code is also important. Be sure to provide comments in your code explaining what the code is doing and why.

Review

In Project 1 we worked with JEE servlets and Java Server Pages using the Glassfish application server. We worked with mobile device awareness and the Model View Controller design pattern (MVC).

In this project we will be working at a lower level. That is, we will not have the Glassfish runtime environment to rely on. You may, however, continue to use Netbeans for all of this work. In this project, we will be programming with UDP and TCP sockets. We will not be building web applications. In Netbeans, create projects that are Java Applications – not web applications.

Discussion of UDP and TCP

In this project you will work with both UDP and TCP over IP. UDP is a simpler protocol than TCP and therefore may require more work from the application programmer. As was mentioned, UDP is used by the Domain Name System (DNS). DNS is a system that allows us to quickly convert a user-friendly name, such as www.cmu.edu, into an IP address. The IP address is then used to make requests.

TCP presents the application programmer with a stream abstraction and is busier behind the scenes. TCP, unlike UDP, tries its best to make sure packets are delivered to the recipient. The underlying network we are using may, on occasion, drop packets. So, how can TCP provide for reliable delivery of packets? TCP uses the fundamental principal of "positive acknowledgement with retransmission".

A simplified example of "positive acknowledgement with retransmission" looks like the following. In this scenario, no packets are lost.[1]

[1] These notes are adapted from "Internetworking with TCP/IP, Volume I: Principles, Protocols and Architecture" by Douglas E. Comer

Positive acknowledgement with retransmission

```
Sender                          Service
=====================================================

  Send packet 1
  Start timer
                                  Receive packet 1
                                  Send ACK for packet 1
  Receive ACK 1
  Cancel timer

  Send packet 2
  Start timer
                                  Receive packet 2
                                  Send ACK for packet 2
  Receive ACK 2
  Cancel timer
```

Here is an example where the first packet is lost.

```
Sender                          Service
=====================================================

  Send packet 1
  Start timer
                 Packet lost


  Time expires
  Send packet 1
  Start timer
                                 Receive packet 1
                                 Send ACK for packet 1

  Receive ACK 1
  Cancel timer

  Send packet 2
  Start timer
                                 Receive packet 2
                                 Send ACK for packet 2
  Receive ACK 2
  Cancel timer
```

Here is an example where the first ACK is lost.

```
Sender                          Service
=====================================================

  Send packet 1
  Start timer
                                  Receive packet 1
                                  Send ACK 1
                  ACK 1 lost

  Time expires
  Send packet 1
  Start timer
                                  Receive packet 1 a second time
                                  Send ACK for packet 1

  Receive ACK 1
  Cancel timer

  Send packet 2
  Start timer
                                  Receive packet 2
                                  Send ACK for packet 2
  Receive ACK 2
  Cancel timer
```

The acknowledgement may be replaced with the result of the service. Here is another example.

```
Sender                          Service
=====================================================

  Send packet 1
  Start timer
                                  Receive packet 1
                                  Send response
                 Response
                 Lost

  Time expires
  Send packet 1
  Start timer
                                  Receive packet 1 a second time
                                  Send response again

  Receive response
  Cancel timer

  Send packet 2
  Start timer
                                  Receive packet 2
                                  Send response for packet 2
  Receive response
  Cancel timer
```

Task 1

In Figures 4.3 and 4.4 of the Coulouris text, two short programs are presented. UDPClient.java and UDPServer.java communicate by sending and receiving UDP packets. The code for these programs can be found at:

http://www.cdk5.net/wp/extra-material/source-code-for-programs-in-the-book

Make modifications to UDPClient.java and UDPServer.java so that the client may ask the server to perform a lookup operation. The server will maintain a Java TreeMap of city name and location pairs. When a UDP packet arrives containing a city name, the server responds with the city's location. The city's location will be expressed as longitude and latitude coordinates. In my solution, I have added cities to the TreeMap with code like this:

```
cityLocationPairs.put("Pittsburgh,PA","40.440625,-79.995886");
```

The server will be executed first and will announce each visit. An example execution will look like the following:

```
java UDPServer
Handling request for Pittsburgh,PA
Handling request for New York,NY
Was unable to handle a request for pittsburgh,PA
Handling request for Washington,DC
```

The execution of the client program will prompt the user for input and will look like the following three examples:

```
java UDPClient
Enter city name and we will find its coordinates
Pittsburgh,PA
40.440625,-79.995886
```

```
java UDPClient
Enter city name and we will find its coordinates
New York,NY
40.7141667,-74.0063889

java UDPClient
Enter city name and we will find its coordinates
pittsburgh,PA
Could not resolve 'pittsburgh,PA'
```

Note that the server and not the client actually performs the lookup operation. Note too that only one UDP message is sent to the server. The message contains a city name and state abbreviation.

If the city name is not in the table held by the server, the server returns an empty string. In this way, the client can identify when the server was unable to complete the task. In this case, neither the client nor the server will crash.

Your server should be coded to handle a minimum of four cities. The cities you choose are up to you. Your screen shots will show each of these four cities being accessed and will show at least one failure.

UDPClient.java and UDPServer.java will be placed in a project called Project2Task1 and submitted to Blackboard. Note, the client and the server are in the same project. There will be two main routines in this project. Be sure to run the server first.

Task 2

Build a new project called Project2Task2. Modify the UDPServer.java code and create a new Java class called UDPServerThatIgnoresYou.java. Write the new server so that it randomly ignores 90% of requests – a very unreliable server. In other words, the new UDPServerThatIgnoresYou will contain code close to this:

```
// rnd is an object of the Random class
aSocket.receive(request);
if(rnd.nextInt(10) < 9) {
      System.out.println("Got request " +
          new String(request.getData())+
            " but ignoring it.");
      continue;
}
else {
      System.out.println("Got request" +
          new String(request.getData()));
      System.out.println("And making a reply");
}
```

Create a new client called UDPClientWithReliability.java. This new client is a modified UDPClient from Task 1. After a request, it waits only 1 second for a reply. If the reply does not arrive after one second, the client tries again. Unlike many such systems, this one never gives up. The client side UDP receive logic will look something like this.

```
aSocket.setSoTimeout(1000);
aSocket.receive(reply);
```

You will need to research how sockets handle time outs and be sure to see the above discussion on "positive acknowledgement with retransmission".

UDPClientWithReliability.java will have a main routine that asks the user to enter a city name and displays the coordinates provided by the server. There must be no socket code in the main routine. Instead, the main routine will make a call on a routine with the signature

static String getLocation(String city)

All of the socket work (and retry code) will be done within getLocation().

```
//precondition: city has a name in the form “city,state abbreviation”
// postcondition: Longitude and latitude coordinates are returned or
// an empty string is returned if the city is not in our table.
public static String getLocation(String city);
```

All files for this Task should be in a project named Project2Task2. You only need to change the server and implement the getLocation() method to perform retries. This is an example of adding reliability to UDP.

Task 3

This project will be named Project2Task3.

In this task we will make use of the Tiny Encryption Algorithm (TEA). You are not required to understand the underlying mechanics of TEA. You will need to be able to use it in your code. Later in the course, we will discuss TEA and describe it as one of many symmetric key encryption schemes. TEA is well known because of its small size and speed.

In Figure 4.5 and 4.6 of the Coulouris text, two short programs are presented: TCPClient and TCPServer. You can also find these programs at:

http://www.cdk5.net/wp/extra-material/source-code-for-programs-in-the-book

The TCPClient program takes two string arguments: the first is a message to pass and the second is an IP address of the server (e.g. localhost). The server will echo back the message to the client. Before running this example, look closely at how the command line argument list is used. You will need to include localhost on the command line. In Netbeans, command line arguments can be set by choosing Run/Set Project Configuration/Customize.

Make modifications to the TCPClient and TCPServer programs so that spies in the field are able to securely transmit their current location (longitude and latitude) to Intelligence Headquarters (at a fixed location). Intelligence Headquarters is run by Sean Beggs. There are three spies and one spy commander as listed here:

User-id	password	title	location
jamesb	james	spy	long, lat, alt
joem	joe	spy	long, lat, alt

mikem	mike	spy	long, lat, alt
seanb	sean	Commander	-79.945289°, 40.44431°, 0

The spies are required to inform Sean of their locations as they move about the world (on super secret missions). Sean uses Google Earth to view the locations of his spies. The spies communicate over a channel encrypted using TEA. TEA is a symmetric key encryption algorithm and so Sean has provided his spies with the symmetric key before they left Hamburg Hall on their missions. Of course, Sean's software knows the ID and password of each spy. So, while TEA is used for encryption, authentication is provided by the user name and password.

Name these two new programs TCPSpyUsingTEAandPasswords.java and TCPSpyCommanderUsingTEAandPasswords.java. The first is a TCP client used by each spy in the field. The second is a TCP server used by Spy Commander Beggs.

Here is an example execution of the server on Sean's machine.

```
java  TCPSpyCommanderUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretsodon'ttellanyone
Waiting for spies to visit...
Got visit 1 from Mike
Got visit 2 from Joe
Got visit 3 from James
Got visit 4 illegal symmetric key used.
Got visit 5 from James. Illegal Password attempt.
```

Here is an example execution of the client on Mike's machine.

```
java  TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretsodon'ttellanyone
Enter your ID: mikem
```

Enter your Password: mike
Enter your location: -79.956264,40.441068,0.00000
Thank you. Your location was securely transmitted to Intelligence Headquarters.

Here is an example execution of the client on Joe's machine.
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretsodon'ttellanyone
Enter your ID: joem
Enter your Password: joe
Enter your location: -79.945389,40.444216,0.00000
Thank you. Your location was securely transmitted to Intelligence Headquarters.

Here is an example execution of the client on James Bond's machine.
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
thisissecretsodon'ttellanyone
Enter your ID: jamesb
Enter your Password: james
Enter your location: -79.940450,40.437394,0.00000
Thank you. Your location was securely transmitted to Intelligence Headquarters.

Here is an example execution of the client by Mallory. (She broke into Joe's office and ran his copy of the client.)
java TCPSpyUsingTEAandPasswords
Enter symmetric key for TEA (taking first sixteen bytes):
IBetTheyUseThisKeyAsTheSecret
Enter your ID: joem
Enter your Password: sesame
Enter your location: -79.940450,40.437394,0.00000

Some exception is thrown on client (The server ignores this request after detecting the use of a bad symmetric key and the server notifies Sean.)

Here is an example execution of the client by James Bond (who forgot his password).

java TCPSpyUsingTEAandPasswords

Enter symmetric key for TEA (taking first sixteen bytes):

thisissecretsodon'ttellanyone

Enter your ID: jamesb

Enter your Password: jimmy

Enter your location: -79.940450,40.437394,0.00000

Not a valid user-id or password.

After each visit by an authenticated spy, the server writes a file called SecretAgents.kml to Sean's desktop. Here is a copy of a typical KML file. This file can be loaded into Google Earth.

SecretAgents.kml

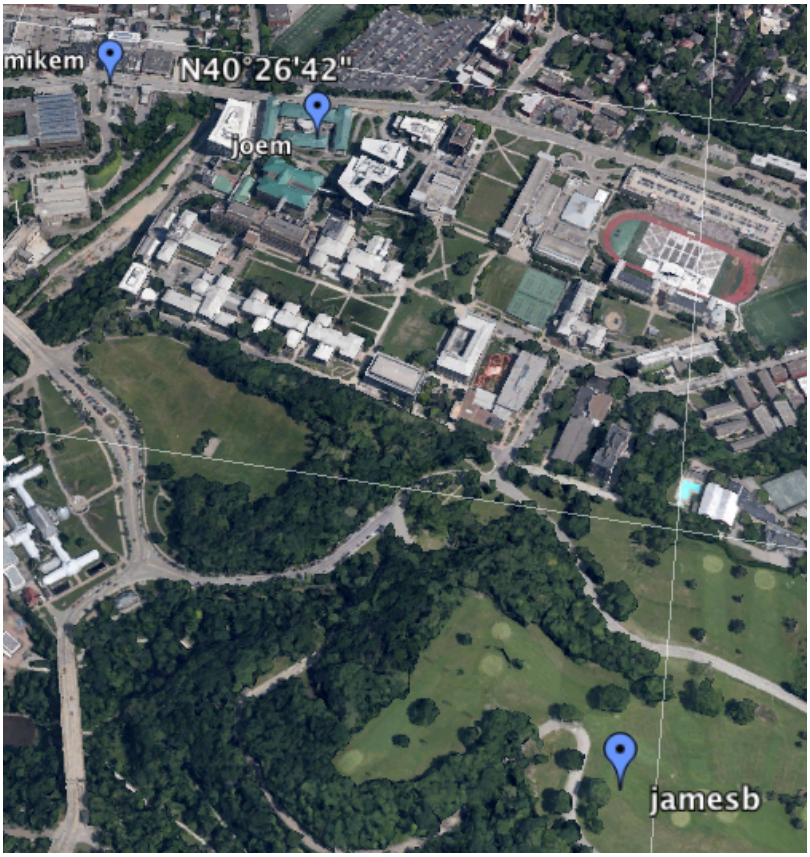
```
<?xml version="1.0" encoding="UTF-8" ?>
<kml xmlns="http://earth.google.com/kml/2.2"
><Document>
<Style id="style1">
<IconStyle>
<Icon>
<href>http://maps.gstatic.com/intl/en_ALL/mapfiles/ms/micons/blue-dot.png</href>
</Icon>
</IconStyle>
</Style>
<Placemark>
  <name>seanb</name>
  <description>Spy Commander</description>
  <styleUrl>#style1</styleUrl>
  <Point>
    <coordinates>-79.945289,40.44431,0.00000</coordinates>
  </Point>
</Placemark>
<Placemark>
  <name>jamesb</name>
  <description>Spy</description>
  <styleUrl>#style1</styleUrl>
  <Point>
    <coordinates>-79.940450,40.437394,0.0000</coordinates>
  </Point>
</Placemark>
```

```
<Placemark>
  <name>joem</name>
  <description>Spy</description>
  <styleUrl>#style1</styleUrl>
  <Point>
    <coordinates>-79.945389,40.444216,0.00000</coordinates>
  </Point>
</Placemark>

<Placemark>
  <name>mikem</name>
  <description>Spy</description>
  <styleUrl>#style1</styleUrl>
  <Point>
   <coordinates>-79.948460,40.444501,0.00000</coordinates>
  </Point>
</Placemark>

</Document>
</kml>
```

When loaded, SecretAgents.kml looks similar to this in Google Earth.



Note that Joe is at work in Hamburg Hall, Mike is hanging out at Starbucks and James Bond is golfing. To see where Sean is located, load the KML file above into Google Earth.

You are required to rewrite the entire file (SecretAgents.kml) after each visit from a spy. The file always contains data on all three spies and the spy commader. This means that you need to maintain the state on the server for each spy. This would include location data, user ID, password, and a title for display on Google Earth (see the Description element in the KML file).

If a visitor (spy) does not have the correct ID or password, no change will be made to the SecretAgents.kml file. The server will send a message to the client saying illegal ID or password.

If a visitor (evil spy) enters an illegal symmetric key, the server will detect that and close the socket. How the client behaves at this point is of no real concern. The server should not deal at all with anyone with an illegal symmetric key. One way you might detect the use of this attack is to check if the decrypted data is properly formatted and standard ASCII. It would be a mistake to have the symmetric key stored

in the Java code on the client. That is, you can't simply test that the user entered symmetric key against a key stored in the client side code.

You may assume that the location data is accurate and well formed. That is, you do not have to validate the longitude, latitude, or altitude. The spies are always careful to enter these data correctly.

Initially, before any spy has communicated with the server using TEA over TCP, all of the spies have their initial state stored in memory objects on the server. How you do this is of your own design. Each spy is initially located in Hamburg Hall (see Joe's coordinates in the KML file).

As soon as the first spy visits using TEA and TCP, the SecretAgents.kml file is re-written with that spy's new location. The other values (for the other spies still located in Hamburg Hall) are also re-written to the file. Thus, the file should always have data for all three spies (in Hamburg Hall or not). The KML file only needs to be written. It is read only by Google Earth. The KML file may be written as a single Java String. There is no need for an XML parser, we are only writing an XML string to a file.

From the Spy Commanders point of view, he runs the server and leaves it running all day and all night. On occasion, perhaps every few hours, he loads the SecretAgents.kml file into Google Earth to see where his spies are located. We are not writing an automatic refresh into Google Earth (maybe next term).

See Wikipedia and see the course schedule for a copy of TEA.java (which you may use.) Name this project Project2Task4. It will contain the files:

TCPSpyUsingTEAandPasswords.java and

TCPSpyCommanderUsingTEAandPasswords.java

TEA.java. Other files may be included as needed.

In my solution, since I am reading and writing streams of bytes, I did not use writeUTF() and readUTF(). Instead, I used these methods in DataInputStream and DataOutputStream:

```
public final int read(byte[] b)

public void write(byte[] b)
```

The return value of the read method came in very handy.

Note: You may not assume that the symmetric key entered (by a spy) is valid. That is, you should detect when an invalid key is being used. You may assume that the key that the Spy Commander enters is correct and has been secretly provided to and memorized by each spy. Hint: Postpone this concern until you have the happy case working.

Finally, rather than storing the user id and password in the server side code (insecure if Eve steals a copy of the server source code), store the user id, some cryptographic salt and a hash of the salt plus the password in the code. When authenticating, use the user id to find the salt and hash of salt plus password pair. Hash the salt with the user provided password and check for a match with the stored hash of salt plus password pair. You may use SHA-1 or MD5 as you did in the first project. Hint: Do this last, after everything else is working well.

Final note. It is a bad idea to write your own cryptography software – unless you are an expert with years of experience. We do this exercise only to understand and explore some major issues in cryptography. Or, perhaps, it will spark such a career in computer security.

Questions

Questions should be posted to the Blackboard Discussion Board, under the Project 1 Discussion Forum.

Project 2 Summary

Be sure to review the grading rubric on the schedule. We will use that rubric in evaluating this project. Documentation is always required.

The Netbeans projects will be named as follows:

- Project2Task1
- Project2Task2
- Project2Task3

You should also have three screen shot folders:

- Project2Task1ScreenShots
- Project2Task2ScreenShots
- Project2Task3ScreenShots

For each Netbeans project, File->Export Project->To Zip...each. You must export in this way and NOT just zip the Netbeans project folders. In addition, zip all the Netbeans export zips and the screenshot folders into a folder named with your andrew id.

Zip that folder and submit it to Blackboard.

The submission should be a single zip file.